Hardware Support for Dynamic Languages

Schleuniger, Pascal; Karlsson, Sven; Probst, Christian W.

Published in:
ACACES 2011 Seventh International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems

Publication date:
2011

Hardware Support for Dynamic Languages

Pascal Schleuniger*, Sven Karlsson*,
Christian W. Probst*,[1]

* *DTU Informatics, Technical University of Denmark, Richard Petersens Plads, Building 322, 2800 Kgs. Lyngby, Denmark*

ABSTRACT

In recent years, dynamic programming languages have enjoyed increasing popularity. For example, JavaScript has become one of the most popular programming languages on the web. As the complexity of web applications is growing, compute-intensive workloads are increasingly handed off to the client side. While a lot of effort is put in increasing the performance of web browsers, we aim for multicore systems with dedicated cores to effectively support dynamic languages. We have designed Tinuso, a highly flexible core for experimentation that is optimized for high performance when implemented on FPGA. We composed a scalable multicore configuration where we study how hardware support for software speculation can be used to increase the performance of dynamic languages.

1 Exploiting parallelism in JavaScript

Dynamic languages such as JavaScript, PHP, Python, and Ruby are typically executed on a virtual machine or interpreter. A lot of effort has been invested in increasing the efficiency of those execution platforms, but parallelization remains a huge challenge. Common approaches to exploit parallelism of the program code often include software speculation. For JavaScript this is a challenging task:

- JavaScript is single threaded by design; missing internal locking mechanisms make it difficult to apply parallelism.
- JavaScript is run via interpretation or by dynamic just-in-time compilation, hence applying intensive code analysis as done by static compilers to exploit parallelism is too expensive. Parallelization algorithms are applied dynamically and therefore need to be as simple as possible.

However, JavaScript web-applications are often executed event driven and Fortuna et al. [FACE10] show that parallelization of events has the potential for significant speedups.

[1]E-mail: {pass,ska,probst}@imm.dtu.dk

Speculative fetching of page data and pre-computing its layout as applied by the Crom speculation engine [MEHL10] makes subsequent page loads faster.

Parallelism of JavaScript code can be exploited using thread level speculation, *TLS*. Martinsen and Grahn's [MG10] TLS method is based on the assumption that function calls that return a null value are likely to be independent from the rest of the program. Once a function returns a null value it is marked and the next time it is called it is executed concurrently as thread. During the execution, reads and writes to memory are monitored to detect data dependence violations. In case of a conflict, a rollback is performed and the function is re-executed sequentially. Parascript [MHSM11] is a parallelization system that applies an optimistic runtime scheme for identifying parallelizable loops and dynamically generates parallel code. Multiple threads are executed concurrently; dependencies among the threads are monitored using reference counting. Again, when a conflict check fails, the parallel loop execution is rolled back to the checkpoint at the beginning of the loop and the loop is executed sequentially instead.

2 Hardware support for software speculation

We conclude that exploiting parallelism in dynamic programming languages, such as JavaScript, will benefit significantly from software speculation. Accordingly, we propose multicore systems with dedicated cores that provide hardware support for software speculation. We approach hardware support for software speculation at the following levels:

- The hardware shall support predicated instructions that are executed if a specified condition is true, otherwise the instruction is annulled and has no effect. Predicated instructions allow for converting a control dependence into a data dependence, which likely eases code analysis during the parallelization process.

- The Transmetas's Crusoe microprocessor [DGB+03] supports rollback of failed instruction translations. We envision similar hardware support for rollback of incorrectly speculated code. For a subset of the register file, there exist two copies of the registers, a working copy and a shadow copy. When executing speculative code the working copy registers are used to store updates. These updates can, under software control, either be atomically committed or atomically discarded.

- When speculating ambitiously the hardware must be able to ignore exceptions of code sequences that are executed speculatively until we know that such exceptions really occur.

- When executing loop iterations as concurrent threads, the hardware can assist in monitoring data dependence violations and trigger an exception if the speculation fails.

- The hardware shall support pre-fetching of data. This efficiently hides some of the memory access latency.

3 Hardware experimentation platform

We designed Tinuso [SK10], a highly flexible core for experimentation on FPGAs. Tinuso is a statically-scheduled, single-issue, RISC processor architecture. It uses a 8-stage pipeline to attain a high system-clock frequency. Tinuso utilizes block RAMs for both instruction and data cache as well as the register file. We reach a significantly higher system-clock frequency than current production cores by pipelining cache access and execution stage. Predicated instructions are used to circumvent costly pipeline stalls due to branches. The core can be equipped with an optional integer multiplication unit and barrel-shifter.
We implemented a network-on-chip and a network interface to compose multicore systems. The non-blocking, packet-switched network utilizes XY routing. Currently we are implementing a second level cache in the network interface of the individual processor tiles. To efficiently support concurrent execution of threads, hardware assisted cache coherency is required.

4 Conclusion

With growing complexity of web applications, compute-intensive workloads are increasingly handed off to the client side. There is, for example, an increasing need for high performance JavaScript execution. We introduce various approaches to exploit parallelism in JavaScript code and conclude that software speculation is heavily used. We aim for multicore systems with dedicated cores to effectively support dynamic languages. We propose predicated instructions, hardware support for rollback and commit, exception handling, data dependence monitoring and hardware support for pre-fetching data.

References

[DGB+03] J. Dehnert, B. Grant, J. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson. The transmeta code morphing software: using speculation, recovery, and adaptive retranslation to address real-life challenges. In *Proceedings of CGO*, pages 15–24, 2003.

[FACE10] E. Fortuna, O. Anderson, L. Ceze, and S. Eggers. A limit study of javascript parallelism. In *Proceedings of IISWC*, pages 1–10, 2010.

[MEHL10] J. Mickens, J. Elson, J. Howell, and J. Lorch. Crom: Faster web browsing using speculative execution. In *Proceedings of NSDI*, pages 127–142, 2010.

[MG10] J. Martinsen and H. Grahn. An alternative optimization technique for javascript engines. In *Proceedings of MCC*, pages 155–160, 2010.

[MHSM11] M. Mehrara, P. Hsu, M. Samadi, and S. Mahlke. Dynamic parallelization of javascript applications using an ultra-lightweight speculation mechanism. In *Proceedings of HPCA*, pages 87–98, 2011.

[SK10] P. Schhleuniger and S. Karlsson. A processor architecture for a multi-core hardware simulation. In *Proceedings of MCC*, pages 25–30, 2010.